TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

BAKALÁŘSKÁ PRÁCE

Liberec, 2007

Jakub Petkov

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B2612 – Elektrotechnika a infomatika
Studijní obor: 2612R011 – Elektronické informační a řídící systémy

Interaktivní ovládání algoritmů pro slepou separaci signálů v Matlabu

Interactive control of blind source separation algorithms in Matlab

Autor: **Jakub Petkov**

Vedoucí BP: Ing. Zbyněk Koldovský, PhD.

% zadání bakalářské práce

Anotace

Cílem této práce je navržení interaktivního rozhraní (GUI) pro ovládání některých algoritmů slepé separace signálů v Matlabu, dle vzoru FastICA package. Aplikace pracuje se signály uloženými do proměnných v Matlab workspace, umožňuje přehledné zobrazování jak vstupních, tak výstupních dat a vykreslování výstupních charakteristik (ISR matice). Výsledky této práce by měly usnadnit používání algoritmů slepé separace signálů.

Práce je tématicky rozdělena do dvou hlavních částí. Kapitola 2 pojednává o základech teorie slepé separace signálů a popisuje základní metody, které používá. Druhá část (kapitoly 3 a 4) se věnuje tvorbě grafického uživatelského rozhraní v Matlabu a popisuje návrh a obsluhu konkrétního rozhraní pro BSS.

Abstract

The purpose of this work is to design a graphical user interface (GUI) in Matlab to control algorithms of Blind Source Separation, according to the FastICA package. Application uses signals saved into variables in the Matlab workspace, it enables transparent vizualization of input and output data, and of Interference to Signal Ratio matrices (ISR). Results of this work should enable using BSS algorithms for larger group of people.

The work is thematicaly divided into two main parts. Chapter 2 explains basics of Blind Source Separation and describes several methods. The second part, chapters 3 and 4, explains how to create graphical user interfaces in Matlab and provides description of the implementation add manual of GUI for BSS.

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o použití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvěk na úhradu nákladů vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce.

Dne: 18. 5. 2007

Podpis:

Poděkování

Na tomto místě bych chtěl poděkovat všem, kteří se, ať už přímo či nepřímo, podíleli na úspěšném dokončení této práce. Předně ing. Zbyňku Koldovskému, PhD. za motivaci, příkladné vedení a velkou ochotu. Dále svým rodičům za poskytnutí těch nejlepších studijních podmínek. V neposlední řadě také Kristýně Borůvkové za pochopení a podporu.

Obsah přiloženého CD

bakalarska_prace.pdf – elektronická podoba této práce
..\GUI – zdrojový kód všech použitých M-files v kompletním package:

- *GUI.m, GUI.fig* – hlavní okno rozhraní
- *myplot.m, myplot.fig* – vykreslování vícerozměrných signálů
- *showmatrixISR.m* – grafické zobrazování matic v log. měřítku
- *nearest2.m* – pomocný M-file k výpočtu gain matice *G*
- *efica.m*, *ewasobi.m*, *combi.m*, *multicombi.m*, *fcombi.m* – použité algoritmy BSS

..\img – obrazová dokumentace programu

Obsah

1 Úvod

Oddělování nejrůznějších typů zamixovaných dat je problém, jehož řešení nalézá uplatnění ve spoustě rozdílných oborů. O výhodách vyčištěných EEG a EKG záznamů od rušivých vlivů jistě nemusím nikoho přesvědčovat. Stejně tak, jako o oddělení mluveného slova od doprovodné hudby (ideální podmínky pro počítačové zpracování řeči). Hudebně zainteresovaní jedinci pak jistě ocení možnost poslechnout si jednotlivé nástroje zvlášť, odseparované ze záznamu celého hudebního tělesa. Sami si jistě dovedete představit celou řadu podobných situací.

Algoritmy, které se tuto problematiku pokouší více či méně úspěšně řešit, jsou však z velké části dostupné pouze úzkému okruhu lidí s určitou úrovní počítačové gramotnosti. Bylo by jistě zajímavé pokusit se zjednodušit práci s algoritmy slepé separace tak, aby mohl být kladen důraz na pohodlí a jednoduchost ovládání. První kroky již byly v tomto směru podniknuty, FastICA package obsahuje grafické uživatelské rozhraní pro obsluhu algoritmu FastICA. V této práci jsem se pokusil na tento projekt navázat a vytvořit tak kompaktní rozhraní pro ovládání algoritmů EFICA, WASOBI, COMBI, MULTI-COMBI a FCOMBI.

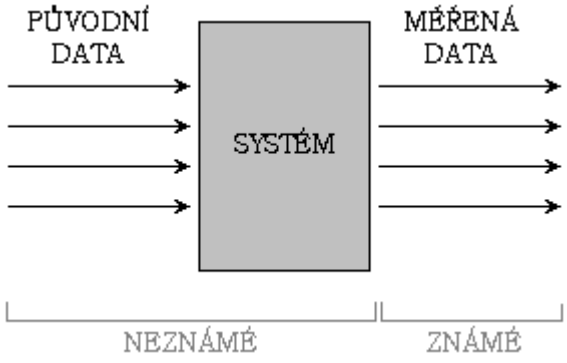
2 Základní principy slepé separace signálů (BSS)

Slepá separace signálů (BSS – blind source separation) je obor zabývající se oddělováním zamixovaných signálů či měřených dat. I za poměrně nepříznivých podmínek (případ, kdy známe pouze původní směs signálů) dokážeme užitím některé z jejích konkrétních metod odseparovat všechny její dílčí složky. Názorným příkladem je tzv. coctail-party problém. Vychází z představy, že chceme pořídit zvukový záznam coctail-party, kde současně hovoří větší počet lidí, přičemž nás zajímají pouze konkrétní promluvy jednotlivců. Celou oblast snímáme několika mikrofony umístěnými v různých částech místnosti. Tím získáme určitý počet odlišných záznamů, ze kterých jsme, za pomoci slepé separace signálů, schopni oddělit hovory jednotlivých účastníků cocktail-party. Obdobným způsobem pak slepá separace signálů nalézá uplatnění např. v telekomunikaci, zpracování biomedicinských dat, zpracování audio signálů, geoseismických dat apod.

2.1 Základní pojmy

2.2.1 Popis problému

Základní logika chápání problému slepé separace signálů je patrná z (obr. 2.1). Pojem „slepá“ používáme z toho důvodu, že při separaci signálů neznáme původní data ani systém a přesto se snažíme získat co nejvíce informací o původních datech a o systému, resp. o jeho popisu.



Obrázek 2.1: Schéma mixovacího procesu

Měřenými daty rozumíme zpravidla vícerozměrný signál, kde počet jeho složek odpovídá počtu senzorů, kterými byl signál nasnímán. Původní data jsou opět vícerozměrným signálem, přičemž každá jeho složka odpovídá jednotlivému zdroji

jednorozměrného signálu produkovaného zdrojem (lidský hlas, hudba, siréna...). Budeme předpokládat, že rozměr měřených dat bude vždy minimálně roven rozměru dat původních. Jinými slovy, pokud budeme chtít pracovat se čtyřmi zamixovanými zdroji zvuku (obr. 2.1), musíme k jejich snímání použít minimálně čtyřech senzorů.

Při práci s naměřenými daty používáme jejich zápis v maticové podobě, kde řádky matice představují signály nasnímané jednotlivými senzory. Sloupce této matice nazýváme „sample“, čili vzorek. Naměřená data zpravidla označujeme X. Pro vstupní signály platí stejná struktura, avšak označujeme je S.

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mN} \end{pmatrix} \qquad S = \begin{pmatrix} s_{11} & \cdots & s_{1N} \\ \vdots & \ddots & \vdots \\ s_{n1} & \cdots & s_{nN} \end{pmatrix}$$

2.1.2 Lineární model bez paměti

Abychom mohli se signály začít pracovat, musíme nejprve popsat systém, tzn. připodobnit jej k některému konkrétnímu modelu. Nejjednodušším modelem je lineární model bez paměti (linear instantaneous model). Pro něj platí následující předpis

$$X = A \cdot S \tag{2.1}$$

kde X představuje naměřená data, S skutečnou podobu původního signálu a A mixovací matici

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

popisující parametry systému, ve kterém dochází k zamixování signálů.

Jednoduchou matematickou úpravou dále získáme vztah pro přepočet skutečných dat

$$S = A^{-1} \cdot X \tag{2.2}$$

Ve vztazích (2.1) a (2.2) pracujeme s daty dle (obr. 2.1). Pokud si označíme počet původních signálů jako n a počet měřených signálů jako m (jak je patrno již v zápisu matic v kapitole 2.2.1), pak vztah (2.1) platí pro každý jeden vzorek (sample):

$s \ldots$ vektor $\ldots \quad n \times 1 \quad \ldots \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix}$ $\qquad$ $x \ldots$ vektor $\ldots \quad m \times 1 \quad \ldots \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}$

$A \ldots$ matice $\ldots \quad m \times n \quad \ldots \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$ a můžeme psát $x = A \cdot s$.

Pro N vzorků potom analogicky

$S \ldots$ matice $\ldots \quad n \times N$ $\qquad$ $X \ldots$ matice $\ldots \quad m \times N$

Podle vztahů mezi rozměry m a n můžeme rozdělit lineární modely na tři základní skupiny.

První skupinou je model overdetermined ($m > n$). V takovém případě máme k dispozici signály z většího množství senzorů, než je potřeba; $m - n$ signálů je nadbytečných.

Druhým typem je model underdetermined ($m < n$). Jak lze tušit, bez dalších speciálních vlastností již u tohoto modelu nelze přesně rekonstruovat původní podobu signálů. Hodnost matice[1] A je v takovém případě $h(A) < n$. Tento model se řeší metodou pseudoinverze (MMSE separace).

Posledním lineárním modelem bez paměti je model regulární ($m = n$). Počet původních signálů je roven počtu signálů měřených. Předpokládáme, že matice A je maticí regulární[2].

2.1.3 Náhodná veličina

Za náhodnou veličinu X považujeme proměnnou, jež nabývá hodnot podle pravděpodobnostního rozložení. Popsat ji můžeme nejlépe distribuční funkcí F (cdf – cumulative distribution function). Ta nabývá pouze hodnot z intervalu $\langle 0,1 \rangle$, je vždy rostoucí, v limitě je rovna okrajovým bodům zmíněného intervalu.

Pokud je F absolutně spojitá, můžeme zavést pojem pravděpodobnostní hustota f(x). Je to nezáporná reálná funkce taková, že pro všechna reálná x se dá distribuční funkce $F(x)$ náhodné veličiny vyjádřit ve tvaru

$$F(x) = \int_{-\infty}^{x} f(t)\,dt \quad . \tag{2.3}$$

1 **hodnost matice** – číslo, které udává počet lineárně nezávislých řádků matice; platí, že maximální počet lineárně nezávislých sloupců je roven maximálnímu počtu lineárně nezávislých řádků matice

2 **regulární matice** – čtvercová matice, jejíž det. je nenulový, žádný řádek není lineární kombinací jiného, její řádky i sloupce jsou lin. nezávislé, hodnost takové matice o velikosti $m \times m$ je právě m

Vlastností každé hustoty pravděpodobnosti je

$$\int_{-\infty}^{\infty} f(x)dx = 1 \tag{2.4}$$

čili celá plocha pod křivkou $f(x)$ je rovna 1. Dále platí

$$f(x) = \frac{dF(x)}{dx} \tag{2.5}$$

Střední hodnota náhodné veličiny je definována vztahem

$$E[X] = \int_{-\infty}^{\infty} x f(x)dx \tag{2.6}$$

a představuje těžiště rozdělení náhodné veličiny.

Rozptyl náhodné veličiny je střední hodnota kvadrátů odchylek od střední hodnoty; pro spojitou náhodnou veličinu je definován jako

$$\sigma^2 = \int_{-\infty}^{\infty} [x - E(X)]^2 f(x)dx = \int_{-\infty}^{\infty} x^2 f(x)dx - [E(X)]^2 \quad . \tag{2.7}$$

Náhodná veličina s normálním (Gaussovým) rozdělením má rozložení pravděpodobnosti popsané vztahem

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{2.8}$$

kde σ^2 představuje rozptyl náhodné veličiny. Gaussovo rozdělení je jedno z nejdůležitějších rozdělení pravděpodobnosti spojité náhodné veličiny, hustota pravděpodobnosti je symetrická, „zvonovitá“.

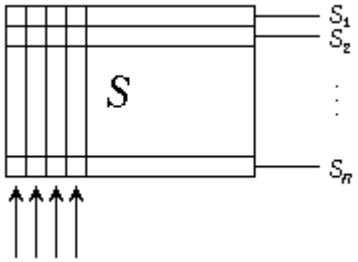
2.2 Analýza nezávislých komponent (ICA)

Analýza nezávislých komponent (Independent Components Analysis) je jednou z nejznámějších metod pro slepou separaci signálů (BSS). Jejím hlavním principem je předpoklad nezávislosti jednotlivých složek původních signálů a na základě toho provést separaci takovou lineární transformací, abychom získali statisticky nezávislé signály. Signály jsou nezávislé, pokud mají nulovou vzájemnou informaci. Ekvivalentním požadavkem je, aby součet entropií transformovaných signálů byl co nejmenší. Entropie je míra neurčitosti náhodné proměnné a je definována jako

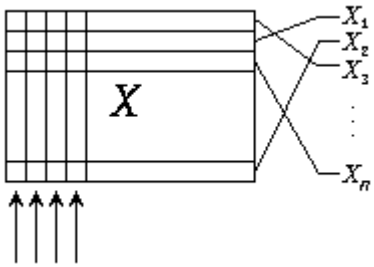
$$H(x) = E[\ln f(x)] = \int_{-\infty}^{\infty} \ln\big(f(x)\big) f(x)\, dx \quad . \tag{2.9}$$

2.2.1 Modely signálů

Pro analýzu nezávislých komponent platí několik předpokladů, které musí vstupní signály splňovat. Původní signály jsou chápány jako nezávislé realizace nezávislých náhodných veličin (obr. 2.2). Nezávislými náhodnými veličinami rozumíme řádky matice *S*, tedy jednotlivé původní signály $S_1, \ldots, S_n$. Za nezávislé realizace považujeme konkrétní samply, přičemž platí, že každý sample je nezávislý na ostatních. Jedná se však pouze o velice zjednodušenou představu, ve skutečnosti tento předpoklad nemusí signály *S* splňovat.



Obrázek 2.2: Původní data



Obrázek 2.3: Zamixovaná data

Pokud dojde k zamixování vstupních signálů (původních dat) mixovací maticí *A*, získáme data *X*. Jak je patrné z (obr. 2.3), řádky matice *X* jsou dle vnitřní struktury matice *A* zamíchané, vzniká mezi nimi určitá závislost. Jednotlivé samply však zůstávají i nadále vzájemně nezávislé.

2.2.2 Základní úloha

Vyjdeme-li z předpokladu, který nám nabízí (2.2), cílem metody řešení separace by měl být co nejpřesnější odhad matice A^{-1}. Odhad demixovací matice A^{-1} označíme *W*. Pro odhadovanou podobu skutečných dat pak můžeme psát

$$\hat{S} = W \cdot X \tag{2.10}$$

Dosazením za $X = A \cdot S$ získáme vztah

$$\hat{S} = W \cdot A \cdot S \quad .$$

Pokud jej přepíšeme ve tvaru

$$\hat{S} = G \cdot S \qquad \text{kde} \quad G = W \cdot A \tag{2.11}$$

zavedeme tak nový pojem, tedy gain matici značenou *G*. Pokud jsou data $\hat{S}$ separovaná dokonale, pak podobu *G* lze zapsat jako součin dvou matic, a to:

Diagonální
: Změní pouze „hlasitost" signálů (pouze roznásobí řádky matice *S*). Tento proces nazýváme „změna škály". Signály v takovém případě zůstávají nezávislé.

Permutační
: Permutační matice změní pořadí výstupních signálů.

Před samotnou aplikací metody ICA bychom měli počítat s jistými předpoklady. Gaussovo (normální) rozložení smí mít pouze nejvýše jeden z původních signálů. Signály, se kterými pracujeme musí mít nulovou střední hodnotu a jednotkový rozptyl, musí být statisticky nezávislé a ve zjednodušené podobě budeme používat nejjednodušší, tedy lineární model.

2.2.3 Postup separace metodou ICA

Princip separace metodou ICA spočívá v minimalizaci vzájemné informace. První fáze výpočtu se nazývá preprocesing. Není nutně nedílnou součástí separace, avšak jeho použití má praktický význam – předzpracovaná data výrazně zkracují dobu potřebnou k samotné separaci. Cílem preprocesingu je splnění nutné podmínky pro nezávislost. Vychází z předpokladu, že pokud jsou dva signály vzájemně nezávislé, musí být i nekorelované. Kovariance dvou signálů je definována vztahem

$$Cov(X_1, X_2) = E[(X_1 - EX_1)(X_2 - EX_2)] \tag{2.12}$$

Pokud uvažujeme střední hodnotu všech signálů rovnou 0, pak můžeme psát

$$Cov(X_1, X_2) = E[X_1, X_2] \tag{2.13}$$

Odhad kovarianční matic spočteme

$$C = \frac{X \cdot X^T}{N} \tag{2.14}$$

Ta v každé své *kl*-té položce obsahuje odhad kovariance definovaný jako

$$\frac{1}{N}\sum_{i=1}^{N} X_k X_l \overset{N\to+\infty}{\to} E[X_k, X_l] \tag{2.15}$$

Platí, že pokud jsou data nekorelovaná, pak matice C je diagonální ($C = I$). Následně provedeme dekorelaci. Zavedeme dekorelovaná data Z

$$Z = C^{-\frac{1}{2}} \cdot X \tag{2.16}$$

kde $C^{-1/2}$ je matice splňující podmínku $C^{-\frac{1}{2}} \cdot C^{-\frac{1}{2}} = C^{-1}$. Pokud je C symetrická a pozitivně definitní, pak $C^{-1/2}$ existuje. Dále platí, že

$$\frac{Z \cdot Z^T}{N} = \frac{C^{-\frac{1}{2}} X X^T (C^{-\frac{1}{2}})^T}{N} = C^{-\frac{1}{2}} C C^{-\frac{1}{2}} = I \tag{2.17}$$

takže signály představované maticí Z jsou nekorelované s jednotkovým rozptylem.

Výhodou takto předzpracovaných dat je, že pro libovolný jednotkový vektor w ($\|w\|^2 = 1 = w^T \cdot w$) platí

$$w^T \cdot Z \Rightarrow \frac{w^T Z \left(w^T Z\right)^T}{N} = \frac{w^T Z Z^T w}{N} = w^T I\, w = w^T w = 1 \tag{2.18}$$

což nám umožňuje rychlou orientaci v prostoru signálu s jednotkovým rozptylem. Analogicky dle vztahu (2.18) platí

$$U \cdot Z \Rightarrow \frac{U Z (U Z)^T}{N} = \frac{U Z Z^T U^T}{N} = U U^T I = U U^T = I \tag{2.19}$$

pro $U = \begin{pmatrix} w_1^T \\ \vdots \\ w_n^T \end{pmatrix}$ ortogonální, $U U^T = U^T U = I$, tzn. hledáme ortogonální matici U tak, aby $U \cdot Z$ byly nezávislé

$$U = arg\ min\, I(UZ) = arg\ min \sum_{i=1}^{n} H\left((UZ)i\right) \tag{2.20}$$

kde n je rozměr matice U a $H((UZ)i)$ entropie i-tého signálu.

Posledním krokem je aproximace entropie nelineární nekvadratickou funkcí G (např. x^4 apod.).

$$H(w_i^T) \doteq E[G(w_i^T Z)] \tag{2.21}$$

2.3 Interference to signal ratio (ISR)

Abychom měli představu o přesnosti provedené separace, a tedy i důvěryhodnosti získaných výstupních signálů, potřebujeme zavést určité kritérium, které bude tuto skutečnost vyjadřovat. Pro *i*-tý odhadnutý signál platí

$$\hat{S}_i = G_{i1} S_1 + G_{i2} S_2 + \ldots + G_{id} S_d \tag{2.22}$$

kde G představuje gain matici definovanou vztahem (2.11). V ideálním případě je G jednotková nebo diagonální, popřípadě permutační. Pro *i*-tý odhadnutý signál definujeme *i*-tou složku vektoru *isr*

$$isr_i = \frac{G_{i1}^2 + \ldots + G_{id}^2 - G_{ii}^2}{G_{ii}^2} \tag{2.23}$$

za předpokladu, že G je diagonální (obnovené pořadí). Tato hodnota představuje poměr energie interference (ostatních signálů) a energie *i*-tého signálu (Interference-to-Signal Ratio) v separovaném signálu. V ideálním případě $isr_i = 0$. V souvislosti se složkami vektoru *isr* můžeme také definovat jednotlivé složky vektoru *sir* (Signal to Interference Ratio). Ty určíme jednoduchým vztahem

$$sir_i = \frac{1}{isr_i} \quad . \tag{2.24}$$

Ideálním případem je situace, kdy $sir_i = +\infty$.

Dále definujeme matici *ISR*, která detailněji popisuje reziduální interferenci mezi odhadnutými signály.

$$ISR_{ij} = \frac{G_{ij}^2}{G_{ii}^2} \qquad \text{pro} \quad i \neq j \tag{2.25}$$

$$ISR_{ij} = 0 \qquad \text{pro} \quad i = j \tag{2.26}$$

Není-li G přibližně diagonální (k této situaci dochází téměř vždy), musíme nejprve upravit pořadí jednotlivých řádků tak, aby byla co nejvíce podobná diagonální. Snažíme se najít takovou podobnost, aby byl součet *ISR* signálů co nejmenší.

2.4 Konkrétní metody slepé separace signálů

Metod používaných pro slepou separaci existuje celá řada. Uvádím zde podrobněji algoritmus FastICA a algoritmy, kterým slouží navržený GUI jako ovládací panel.

2.4.1 FastICA, EFICA

První skupinu tvoří algoritmy, které pracují na principu ICA. Hledáme takovou transformaci, aby složky signálu byly nezávislé. Platí pravidlo, že „hodně negaussovské" signály jsou dobře odhadnutelné a naopak.

Jedním z nejznámnějších algoritmů patřících do této skupiny je FastICA navržený Aapo Hyvärinenem. Bývá často považován za jeden z nejlepších, zejména pro svou rychlost a variabilitu. Podstatou tohoto algoritmu je vhodná aproximace entropie, která se dá měnit patřičnou volbou nelinearity a rychlý iterační algoritmus pro hledání minima entropie.

Algoritmus EFICA (Efficient Fast ICA) je ve srovnání s FastICA přesnější a stabilnější. Vychází ze symetrické FastICA. Obsahuje test sedlových bodů, který výrazně napomáhá globální konvergenci algoritmu.

Mezi dalšími algoritmy využívajícími základ ICA můžeme například jmenovat JADE nebo Infomax.

2.4.2 WASOBI

Druhou skupinu tvoří algoritmy, které pracují se stacionárními náhodnými gaussovskými procesy (např. gaussovský autoregresní proces). Zde je podmínkou, aby spektra signálů byla odlišná. V jiném případě jsou touto metodou neodseparovatelná.

Vstupními signály tohoto algoritmu jsou data z (obr. 2.2). Z těchto dat odhadneme tzv. cross-korelační matici dle vztahu

$$R(\tau)_{ij} = E[x_i(t) \cdot x_j(t-\tau)] \tag{2.27}$$

kde E představuje střední hodnotu. Hledáme vhodnou matici W tak, aby při součinu Wx platilo $R(\tau)_{ij} = 0$ pro všechna $i \neq j$ a pro $\forall \tau \in \langle 0, ARorder \rangle$.

2.4.3 COMBI, MULTI-COMBI, FCOMBI

Abychom co nejpřesněji odseparovali signály typu EEG, EKG, zvuk apod., můžeme vhodně zkombinovat oba základní druhy BSS algoritmů (uvedené v odstavcích 2.4.1 a 2.4.2). Tato data totiž obsahují dvojí strukturu, a to časovou i prostorovou. Řešení takového problému nabízí algoritmy poslední skupiny, označované též za algoritmy hybridní.

Existuje opět celá řada nejrůznějších algoritmů, např. ThinICA, JADE^{TD} atd. Kombinací metod se známou teoretickou přesností, EFICA a WASOBI, vznikl účinný algoritmus COMBI. Algoritmus načte vstupní data a vyhodnotí, kterou složku je vhodnější separovat tou kterou metodou. ISR matice je v takovém případě odhadovaná přímo ze separovaných signálů. Zobecněním této metody je algoritmus MULTI-COMBI, FCOMBI (Fast COMBI) je rychlejší verzí již zmiňované metody.

3 Grafické uživatelské rozhraní (GUI) pro Matlab

Pro většinu programů navržených pro Matlab plně postačuje jejich „konzolová“ podoba, avšak při jejich tvorbě musíme brát v potaz skutečnost, že je tak zpřístupníme pouze malé skupině lidí. Skupině, která již jisté zkušenosti s Matlabem samotným má. Pokud však chceme, aby se program stal mnohem více „uživatelsky přátelský“ a mohl jej tak používat téměř každý, měli bychom jej prezentovat v grafické podobě, čili jako aplikaci.

3.1 Obecně o GUI

GUI (graphical user interface), tedy grafické uživatelské rozhraní, umožňuje v přehledné grafické podobě zpřístupnit i ty nejsložitější algoritmy lidem s minimálními znalostmi práce s Matlabem. Umožňuje uživateli snadno ovládat workspace způsobem, který je nesrovnatelně snadnější než přímý zápis instrukcí do příkazové řádky Matlabu nebo spouštění dávkových souborů M-file.

Důležité je podotknout, že prostředí Matlab používá tzv. skriptovací jazyk, tudíž nedisponuje žádným překladačem, jaký známe například u Delphi. Matlab nevytváří žádný samostatně spustitelný soubor, aplikace je proto použitelná pouze jako jeho neoddělitelná součást. To pochopitelně přináší jistá omezení, protože uživatelé musí pro běh třeba i jen jediné jednoduché aplikace instalovat takový obsáhlý a ohromný nástroj, jako je Matlab. Naopak za výhodu bychom mohli považovat možnost měnit podmínky chování aplikace přímo za chodu, a to zadáváním nebo změnou potřebných parametrů přímo pomocí příkazové řádky Matlabu. Další výhodou pak bezesporu je – v případě, že námi navrhovaný GUI obsluhuje M-files – snadná aktualizace, oprava či záměna některých celých funkčních částí programu, aniž bychom museli zasahovat do zdrojového kódu aplikace a následně ji znovu překládat.

3.2 Vývojové prostředí Matlabu

Prostředí Matlab disponuje poměrně rozličným množstvím komponent, které zcela pokryjí veškeré požadavky. Mezi těmi základními jmenujme např. *Push Button*, *Radio Button*, *Static Text*... O nich pojednává kapitola 3.2.2.

Při tvorbě GUI můžeme zvolit dva základní postupy, které se od sebe liší jak způsobem vizualizace GUI, tak i využitelností u různých typů aplikací. Rozdíl mezi nimi by se dal nejlépe přirovnat k rozdílu mezi programováním internetových stránek pomocí HTML kódu a v některé z vizuálních aplikací jako je třeba Microsoft Front Page. Více v kapitolách 3.2.3 a 3.2.4.

3.2.1 Proměnné a pracovní prostory

Pro ukládání proměnných a práci s nimi využívá Matlab několik pracovních prostředí (workspace), která obsahují vzájemně neslučitelné zásobníky proměnných. K hlavnímu pracovnímu prostoru máme přístup z příkazové řádky Matlabu. Druhým typem workspace je lokální pracovní prostor jednotlivých funkcí, proměnné v něm uložené označujeme jako lokální. Poslední typ workspace slouží k práci s globálními proměnnými. Takové proměnné jsou jako jediné dostupné z libovolného místa aplikace.

3.2.2 Komponenty

Při tvorbě grafických rozhraní uživateli zajisté přijde vhod široká nabídka nejrůznějších komponent, kterou Matlab nabízí – a bez kterých se architekt GUI rozhodně neobejde. Rozlišujeme několik základních druhů objektů, *figure*, *uicontrol*, *uipanel*, *uibuttongroup*, *axes* a *actxcontrol*. Následující přehled uvádí kompletní seznam komponent, které jsou v Matlabu dostupné. Pro každou skupinu však platí trochu odlišná pravidla, proto u názvu každé komponenty uvádím v závorce její zařazení.

Figure (figure)
: Základní prostředí, na které umisťujeme ostatní druhy komponent. Na tomto místě bych rád připomněl, že Matlab chápe celý GUI jako *figure*.

Push Button (uicontrol)
: Chová se jako běžné tlačítko, jak ho známe třeba z aplikací určených pro systém MS Windows. Hodí se například k potvrzování nastavení, uzavírání oken dialogů, spouštění funkcí, načítání proměnných, volání procedur apod.

Toggle Button (uicontrol)
: Na rozdíl od *Push Button*, *Toggle Button* zůstává po prvním kliknutí stlačený neustále. Funguje tedy jako jakýsi spínač. Pokud je stlačený, hlásí se v režimu „sepnuto“. Do původní pozice (tedy do režimu „rozepnuto“) se navrátí až s dalším kliknutím myší.

Radio Button (uicontrol)
: Používá se jako přepínač mezi více možnostmi. Většinou vystupuje ve skupině, aktivuje se kliknutím myší a platí pravidlo, že pokud potvrdíme volbu jednoho, deaktivují se ostatní *Radio Button* ve skupině.

Check Box (uicontrol)
: Má obdobnou funkci jako *Radio Button*, tedy výběr z několika předem nastavených možností, avšak s jedním malým rozdílem – *Check Box*

umožňuje zvolit libovolný počet možností současně, aniž by se zrušila jakákoliv volba jiná.

Edit Text (uicontrol)

Komponenty *Edit Text* umožňují uživateli zadávat nebo upravovat textové řetězce (string). Slouží primárně k zadávání vstupní hodnoty v podobě textu, pokud chceme zadávat také číselné hodnoty, musíme string nejprve převést na jeho číselný ekvivalent.

Static Text (uicontrol)

Komponenta, která zobrazuje řádky prostého textu. Typickým použitím *Static Textu* je jako označení nebo popisek komponenty jiné. Může také poskytovat instrukce pro uživatele nebo ve spojení se *Slider* zobrazovat nastavené hodnoty. Důležité je mít na paměti, že uživatelé aplikace nemohou (na rozdíl od *Edit Textu*) *Static Text* interaktivně ovládat.

Slider (uicontrol)

Pomocí posuvníků zadáváme programu číselné volitelné vstupy předem určeného rozsahu. Hodnoty nastavujeme buď přímo posunutím jezdce *Slideru* nebo kliknutím na některou ze dvou šipek. Poloha jezdce je přímo úměrná zadávané hodnotě.

List Box (uicontrol)

Zobrazuje seznam položek, ze kterých si můžeme jednu nebo více vybrat. Ekvivalent komponenty *Check Box*, přičemž při větším množství položek je z hlediska úspornosti prostoru pochopitelně výhodnější použít *List Box*.

Pop-Up Menu (uicontrol)

Po kliknutí na šipku zobrazí seznam položek, ze kterých si uživatel může vybrat pouze jednu. Ekvivalent komponenty *Radio Button*, přičemž při větším množství položek je z hlediska ušetření prostoru výhodnější použít *Pop-Up Menu*.

Axes (axes)

Tato komponenta umožňuje zobrazovat jednoduchou grafiku, jako např. grafy a obrázky.

Panel (uipanel)

Přidružují komponenty do uzavřených skupin, chovají se jako jejich parent[3]. Většinou se používají k seskupování objektů se stejnými nebo podobnými vlastnostmi a zajišťují tak lepší orientaci na GUI.

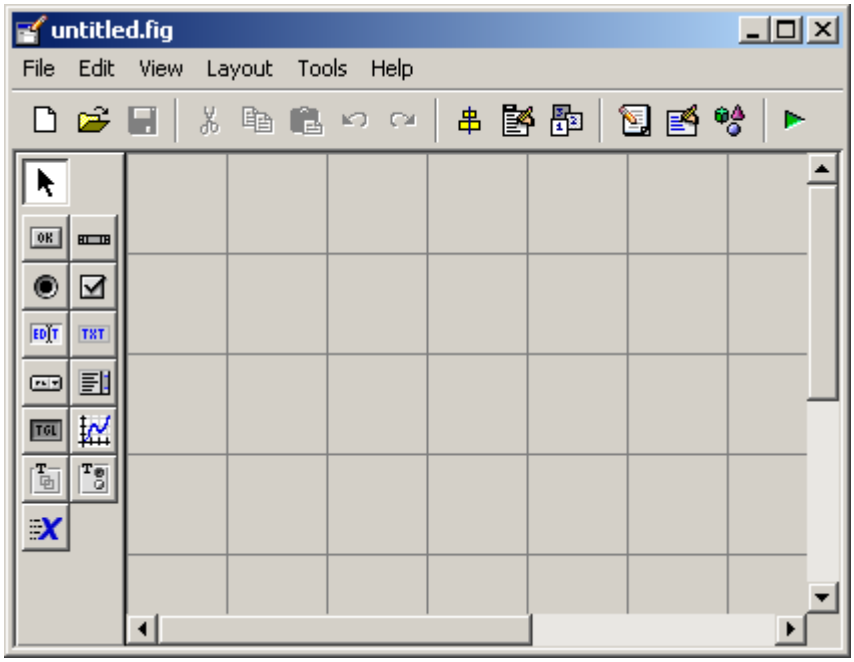
3 **parent** – rodič; objekt obsahující jiný objekt (např. komponentu apod.)

Button Group (uibuttongroup)

Vizuální i funkční podoba *Button Group* je téměř stejná jako *Panel*, s tím rozdílem, že *Button Group* používáme pro správu chování objektů *Toggle Button* a *Radio Button*.

ActiveX Component (actxcontrol)

Umožňuje používat prvky AktiveX přímo ve vašem GUI jako komponentu. Tato možnost je však zpřístupněna pouze uživatelům systém Microsoft Windows. *ActiveX* smí být použita jen jako child[4] samotného objektu *figure*, nelze použít jako child *Panelu* ani *Button Group*. Za použití příslušných *dll* knihoven nám dovolí obsluhovat např. prohlížeč *pdf* souborů, kalendář či Windows Media Player aj. přímo v GUI.



Obrázek 3.1: Matlab graphical user interface development environment

3.2.3 Vizualizace aplikací pomocí GUIDE

Jednou z možností jak vytvořit GUI je použití GUIDE (graphical user interface development environment), tedy vývojové prostředí pro grafické uživatelské rozhraní. Zadáním příkazu *guide* do příkazové řádky Matlabu spustíme základní pracovní oblast s dvěma základními nástrojovými lištami (obr. 3.1).

Na svislé liště nalezneme kompletní přehled všech dostupných komponent,

4 **child** – dítě; objekt podřízený jinému objektu

v prvním sloupci to jsou *Push Button*, *Radio Button*, *Edit Text*, *Pop-Up Menu*, *Toggle Button*, *Panel* a poslední *ActiveX Control.* Druhý sloupec tvoří *Slider*, *Check Box*, *Static Text*, *Listbox*, důležitá komponenta *Axes* a nakonec *Button Group*.

Na vodorovné liště jsou ikony ovlivňující vzhled a funkci GUI umístěny až v druhé polovině. S jejich popisem tedy začnu z pravé strany:

Run
: Tlačítko sloužící ke spouštění aplikace. Důležité k tomu, aby si programátor vyzkoušel funkčnost a vzhled navrhovaného grafického rozhraní.

Object Browser
: Zobrazuje stromovou strukturu rozmístění objektů na *figure*, včetně jejich typů a názvů. Slouží k získání orientace ve větším množství komponent.

Property Inspector
: Kompletní přehled všech vlastností označené komponenty.

M-file Editor
: Kliknutím na ikonu spustíme klasický M-file Editor s vygenerovaným kódem popisujícím právě upravovaný GUI.

Tab Order Editor
: Okno, ve kterém můžeme nastavit pořadí jednotlivých komponent ve vrstvách (z-osa; laicky řečeno, která bude kterou překrývat)

Menu Editor
: Slouží k přidávání nástrojové lišty a roletových nabídek.

Align Objects
: Otevře dialog pro automatické zarovnávání komponent.

Vkládání a umisťování komponent je velice snadné, stejně tak jako nastavování jejich parametrů. To se provádí v Property inspectoru a prakticky veškeré vlastnosti komponent můžeme přizpůsobit právě zde. Co se věrohodnosti návrhu týče, můžeme si být jisti, že pokud budeme používat GUIDE, výsledná podoba aplikace bude téměř totožná s návrhem.

Tento způsob konstrukce grafické podoby aplikace je poměrně rychlý. Komponenty snadno rozmístíme podle svých představ, GUIDE automaticky vygeneruje M-file popisující jejich pozici a vlastnosti.

3.2.4 Vizualizace aplikací přímým programováním

Druhým způsobem jak vytvořit GUI, je pomocí dávkového souboru M-file. V něm

– za použití správné syntaxe – ovládáme přímo konstruktor a sestavujeme tak celkovou podobu aplikace přímým programováním scriptu. Na (obr. 3.2) vidíme zápis generování základního objektu *figure*. Nejprve jej přiřadíme k námi zvolené proměnné, abychom mohli později s komponentou pracovat. V závorce následně uvedeme parametry námi konstruované komponenty. U objektu *figure* rozlišujeme následující vlastnosti:

MenuBar
: Nabývá hodnot 'none' nebo 'figure'. Zobrazuje nebo skrývá nástrojovou lištu, kterou Matlab umisťuje bez rozdílu na každou figure.

Name
: Zadáváme formou textového řetězce. Zobrazí se jako titulek okna *figure.*

NumberTitle
: Nabývá hodnot 'on' nebo 'off'. Povoluje nebo zakazuje číslování *figure.*

Position
: Určuje pozici a velikost *figure*. Tento parametr je 4-prvkovým vektorem o hodnotách [vzdál. od levé strany, vzdál. od spodní hrany, šířka, výška]

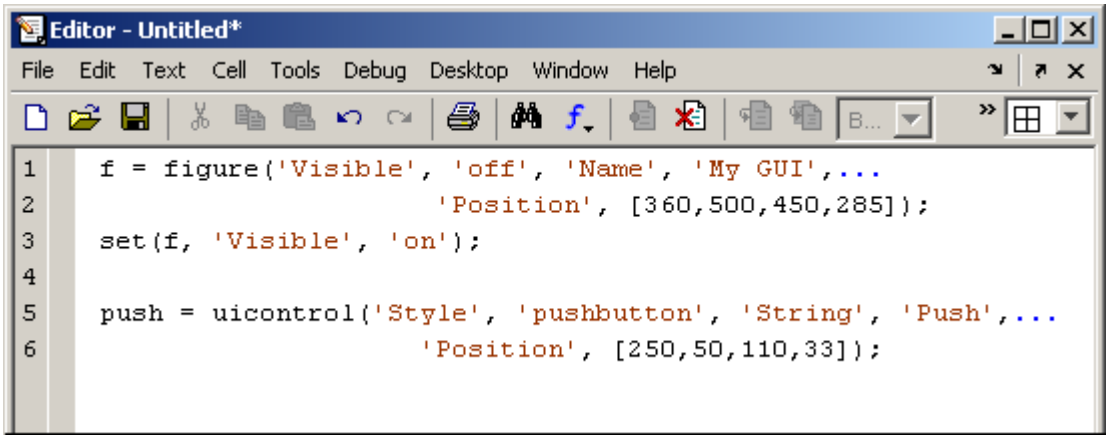
Resize
: Nabývá hodnot 'on' nebo 'off'. Povoluje nebo zakazuje změnu velikosti *figure* tažením myší.

Toolbar
: Vyhrazenými hodnotami jsou: 'none' – *Toolbar* nebude zobrazen, 'auto' – zobrazí *Toolbar*, ale přidáním komponenty na *figure* je opět odstraněn, 'figure' – *Toolbar* zobrazen stále.

Units
: Určuje, v jakých jednotkách budeme na *figure* pracovat. Povoleny jsou:
  'centimetres' – centimetry
  'inches' – palce
  'characters' – definované velikostí defaultního fontu systému; velikost jedné jednotky je rovna šířce znaku 'x'
  'normalized' – normalizované jednotky, kde levý dolní roh představuje bod [0, 0] a pravý horní [1, 1] (při změně velikosti GUI se ve stejném poměru mění i rozměry komponenty)
  'pixels' – nejmenší jednotka digitální rastrové grafiky; představuje jeden bod na monitoru
  'points' – 1 point = 1/72 palce

Visible

Viditelnost *figure*. Nastavíme ji pomocí příznaků 'on' nebo 'off'.



Obrázek 3.2: Ukázka použití konstruktoru

Parametry dalších typů komponent jsou i s jejich popisky a možnými hodnotami přehledně zpracovány v [3], nebudu je zde proto dále rozvádět.

V tomto případě (obr. 3.2) nejprve generujeme *figure*, s parametry *Visible: off, Name: My gui*, umístěna bude v bodě [360, 500] (vztaženo k levému dolnímu rohu obrazovky) s šířkou 450 a délkou 285 bodů, následně ji přiřazujeme proměnné *f*. Příkazem *set* měníme konkrétní hodnoty libovolných vlastností komponenty, v našem případě nastavujeme její viditelnost *Visible: on*.

Další příkaz aktivuje konstruktor a ten vytvoří komponentu *PushButton*, která se bude chovat jako child aktivního objektu *figure*. Bude umístěna v bodě [250, 50], který však již tentokrát bude vztažen k levému dolnímu rohu *figure*, nikoliv obrazovky. Bude mít délku 110 bodů, šířku 33 bodů a popisek *Push*.

Takto můžeme postupně umisťovat komponentu po komponentě na námi zadefinovanou *figure* a získáme tak výslednou podobu grafického rozhraní. Mějme na paměti, že žádnou z výše uvedených vlastností nejsme povinni uvádět. V takovém případě však konstruktor přiřadí komponentě defaultní hodnoty, což nemusí být vždy žádoucí. Pokud zadáme pouze příkaz

```
f = figure;
```

musíme počítat s tím, že vygenerujeme objekt *figure* s největší pravděpodobností zcela nevhodný našim účelům.

Navrhování GUI programováním je poměrně zdlouhavý a pracný postup a může být pro začínající uživatele poměrně složitý. Musíme pro každou komponentu zvlášť vypisovat veškeré její parametry, které od ní budeme později očekávat. Na druhou stranu ale máme plnou kontrolu nad všemi vlastnostmi

jednotlivých vkládaných objektů.

3.2.5 Jakou metodu tedy zvolit?

Každá metoda má svá pro a proti. Každý, kdo navrhuje GUI by měl sám uvážit, jaký způsob mu vyhovuje více. Jak už jsem uvedl výše, GUIDE je výhodné použít převážně při tvorbě rozsáhlejších aplikací.

Pokud běžný uživatel navrhuje jednoduchý GUI k ovládání např. některého ze svých algoritmů, je pro něj asi nejvýhodnější volbou použít GUIDE. Během několika okamžiků tak může vytvořit přehledné rozhraní, které je velice snadné propojit s výkonnou částí programu nebo ji přímo implementovat v GUI.

Chceme-li obsluhovat větší počet M-files nebo potřebujeme zobrazovat pop-up nabídky např. k načítání proměnných, ukládání výstupů programu a zavádění podrobných podmínek jeho chodu, doporučuji zvolit způsob, jaký jsem zvolil ve své práci i já. Hlavní okna s velkým množstvím komponent navrhnout v GUIDE, nechat jej také vygenerovat hrubý základ M-file a dialogová pop-up okna pak naprogramovat přímo.

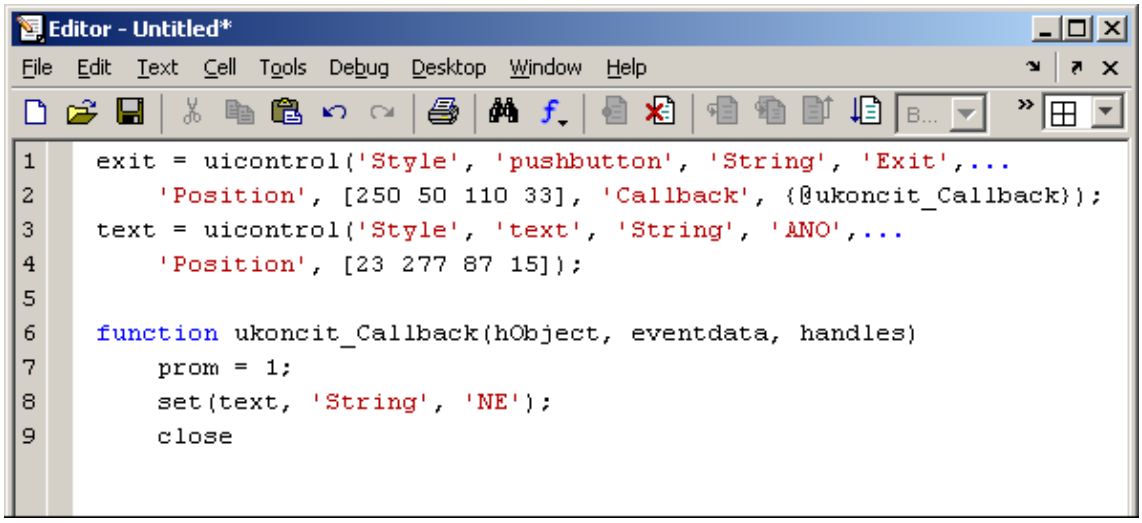
3.3 Základy „oživování“ GUI

3.3.1 Instrukce *callback*

Postupy a metody uvedené v bodě 3.2 slouží pouze k vizualizaci aplikací. Pokud chceme, aby jednotlivé komponenty umístěné na GUI plnily námi požadované funkce, musíme jim přiřadit určitou výkonovou část zdrojového kódu. Toho v prostředí Matlab docílíme pomocí instrukce *callback*. GUIDE generuje předpisy funkcí všech objektů na GUI automaticky ve tvaru

```
function nazevkomponenty_Callback(hObject, eventdata, handles)
```

Všem komponentám pak také přiřadí odkaz na jednotlivé předpisy funkcí. Použijeme-li však ke konstrukci GUI pouze zápis instrukcí uložených do M-file, musíme zadat odkaz ručně, a to specifikací parametru 'callback'. Na (obr. 3.3) vidíme příklad konstrukce tlačítka, kterému přiřadíme funkci s názvem 'ukoncit'. Kliknutím na tlačítko *Exit* se tedy provede řádek po řádku série instrukcí zapsaná pod hlavičkou funkce. Program přiřadí proměnné *prom* hodnotu 1, změní textový řetězec v komponentě *Static text* z hodnoty 'ANO' na hodnotu 'NE' a nakonec uzavře aktivní okno své nadřazené *figure*. Takovýto zápis příkazů však pracuje pouze s lokálními proměnnými funkce. Proměnná *prom* tak zůstává pro další případné použití z workspace nedostupná.



Obrázek 3.3: Příklad přiřazení funkce k objektu

3.3.2 Instrukce *assignin* , *evalin* a *disp*

Práce s proměnnými v Matlab workspace řeší instrukce *assignin*, *evalin* a *disp*. Umožňují obsluhovat workspace přímo z GUI, vkládat, načítat či zobrazovat proměnné, spouštět soubory M-file apod. Základní workspace, hlavní pracovní prostor Matlabu označujeme jako 'base' a workspace funkce, což je pracovní prostor jednotlivých volaných funkcí jako 'caller'.

Potřebujeme-li zálohovat proměnné nebo je zpřístupnit ostatním programům nebo aplikacím, zapíšeme je do workspace příkazem *assignin*. Proměnnou z (obr. 3.3) můžeme použít následujícím způsobem:

```
assignin('base', 'prom_w', prom);
```

Hodnota proměnné *prom* z našeho GUI se tedy takto přiřadí proměnné *prom_w* ve workspace. V opačném případě, pokud chceme proměnnou naopak z workspace načíst, můžeme tak učinit předpisem

```
prom = evalin('base', 'prom_w');
```

Proměnné *prom* takto přiřazujeme hodnotu proměnné *prom_w* z workspace. Pokud chceme pouze zadávat příkazy, které se mají spouštět v příkazovém řádku Matlabu, stačí instrukci *evalin* jen trochu pozměnit

```
evalin('base', 'prom_w');
```

Takto zadaný příkaz např. vypíše ve workspace hodnotu proměnné *prom_w* (ta

musí být také uložena právě tam). Může však dojít k situaci, kdy nepotřebujeme nutně vyměňovat informace mezi GUI a workspace prostřednictvím přiřazování hodnot nově definovaným nebo přepisem již zavedených proměnných. Často stačí danou proměnnou pouze do workspace zobrazit, aby ji měl k dispozici uživatel pouze vizuálně. K tomu použijeme příkaz *disp*:

```
disp(prom);
```

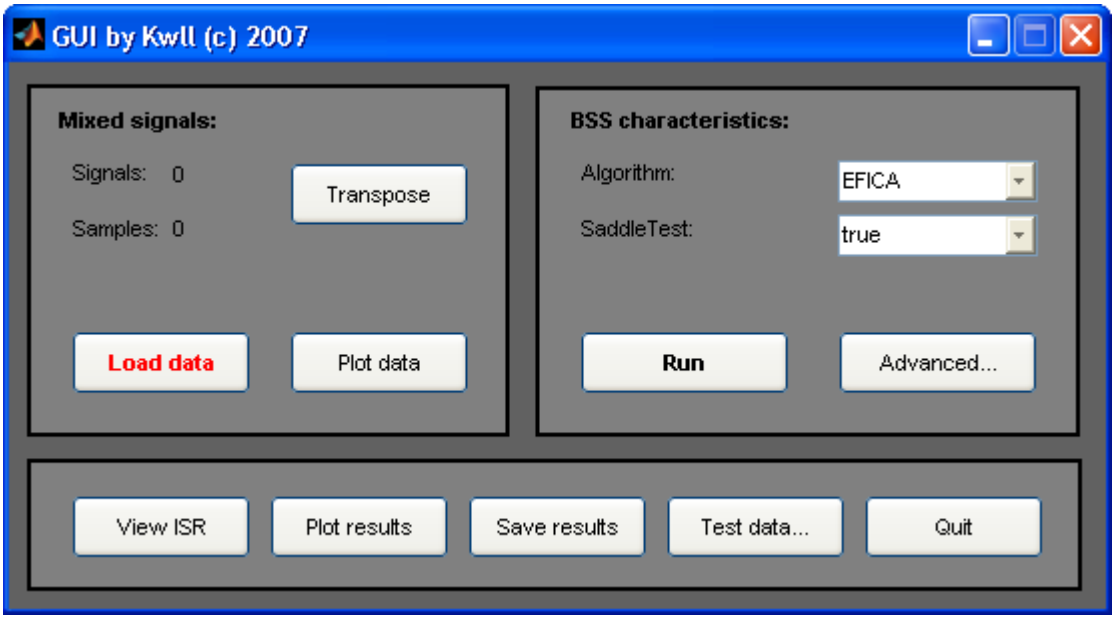
Zobrazená proměnná pak slouží pouze k informativním účelům a nelze tak s ní již dále pracovat.

4 GUI k ovládání algoritmů slepé separace signálů

Hlavním předmětem mé práce je zpracování přehledného rozhraní pro ovládání algoritmů BSS. Navrhl jsem GUI pro obsluhu metod EFICA, WASOBI, COMBI, MULTI-COMBI a FCOMBI. Jejich implementaci načítá GUI z přiložených M-files, které tak spolu s ostatními funkčními částmi programu tvoří celistvý package.

4.1 Základní popis navrženého rozhraní

Grafickou podobu navrženého GUI můžeme vidět na (obr. 4.1). Skládá se ze tří základních částí (panelů), z nichž každý seskupuje určité podobné funkce. Komponenty panelu s označením „Mixed signals:“ slouží k načítání, úpravě a zobrazování vstupních zamixovaných signálů (dat). Panel označený popiskem „BSS characteristics:“ zaopatřuje volbu a obsluhu používaných algoritmů, včetně jejich podrobného nastavení a spouštění. Poslední panel, umístěný pod oběma výše uvedenými, seskupuje komponenty ke zpracování, zobrazení a uložení výsledků aplikovaných metod, dále speciální tlačítka testování dat a ukončení GUI.

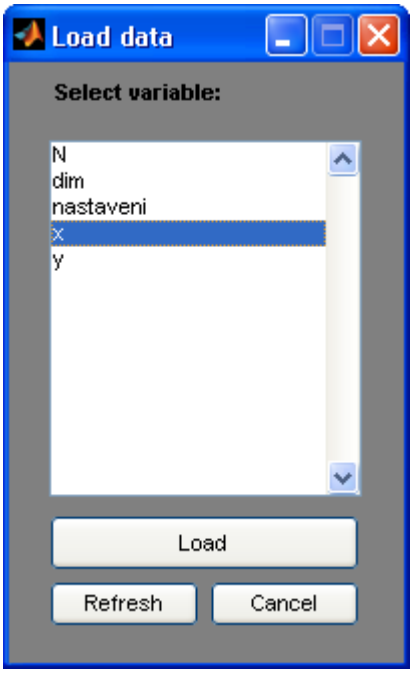


Obrázek 4.1: Základní pohled na navržený GUI

Velikost celého package, včetně pomocných M-file, činí 150 kB. Rozměry hlavního okna jsou 548 x 273 pixelů (cca 14,5 x 7,2 cm). Package spustíme zadáním příkazu *GUI* do příkazové řádky Matlabu.

4.2 Načítání a úprava vstupních dat

Abychom mohli použít některý z dostupných algoritmů, musíme vstupní signály nejprve do aplikace načíst. Stiskem tlačítka označeného popiskem „Load data“ spustíme dialog (obr. 4.2).



Obrázek 4.2: Dialog pro načítání signálů

FastICA package používá pro načítání vstupních signálů figure, ve které musí uživatel ručně zapsat název proměnné, ze které má GUI načíst vstupní data (zamixované signály). Při návrhu vlastního způsobu zadávání vstupních dat jsem vyšel z předpokladu, že veškeré proměnné, které bude chtít uživatel použít, musí být předem uložené ve workspace. GUI (obr. 4.2) je proto při svém spouštění načte a zobrazí je v komponentě *ListBox*. Uživatel si pak jednoduše vybere právě tu, která obsahuje zamixovaná vstupní data a potvrzením tlačítkem „Load“ načte signály do pracovního prostoru aplikace.

Zobrazování proměnných v komponentě *ListBox* vyřešíme následujícími příkazy:

```
var = evalin('base', 'who');
set(hlistbox, 'String', var);
```

Na tomto místě bych se rád zmínil o proměnné *nastaveni* (viz výpis proměnných v *ListBoxu* na obr. 4.2). Tato proměnná se zapíše do workspace při každém

spuštění GUI a po jeho ukončení ji aplikace opět odstraní. Představuje pole složené z osmi na sobě nezávislých řádků, přičemž každý v sobě nese informaci o aktuálním nastavení chodu aplikace. Slouží především pro komunikaci GUI s ostatními částmi celého package, proto důrazně doporučuji uživatelům žádnou z jejích hodnot neměnit, může to zapříčinit nestandardní, zcela neočekávané a především nesprávné chování programu. Stejně tak to platí i pro ostatní proměnné, které si GUI také zavádí sám (např. *N*, *dim* apod.). Tento způsob výměny informací se může zdát na první pohled poněkud „neohrabaný“, avšak díky němu nemusím nijak zásadně přímo zasahovat do naprogramovaných algoritmů. To je výhodou zejména do budoucna, aby byl uživatel schopen aktualizovat verze jednotlivých algoritmů sám.

V panelu označeném jako „Mixed signals“ se po načtení vstupních dat vypíše jejich rozměr s označením „Signals:“ pro počet signálů a „Samples:“ pro počet jejich vzorků. GUI, stejně tak jako algoritmy, které ovládá, pracuje s daty uloženými v řádkových vektorech. Může se však stát, že uživatel má k dispozici pouze data ve vektorech sloupcových. Při načítání proměnných bychom proto měli dávat pozor a sledovat, zda hodnoty označené jako „Signals“ a „Samples“ odpovídají skutečnosti. Pokud tomu tak není, stiskem tlačítka „Transpose“ umístěném na tomtéž panelu, můžeme matici vstupních signálů snadno transponovat.

O zobrazování vstupů (tlačítko „Plot data“) se zmíním později, a to v kapitole 4.5.

4.3 Nastavení jednotlivých algoritmů

Jak již bylo uvedeno výše, volba metody i veškeré ovládací prvky včetně nastavování parametrů a spouštění konkrétních metod nalezneme v panelu označeném jako „BSS characteristics“. Algoritmus zvolíme kliknutím na roletovou nabídku (*Pop-Up Menu*) označenou popiskem „Algorithm:“. Pro každou metodu však platí odlišná pravidla, proto si jejich nastavení probereme zvlášť.

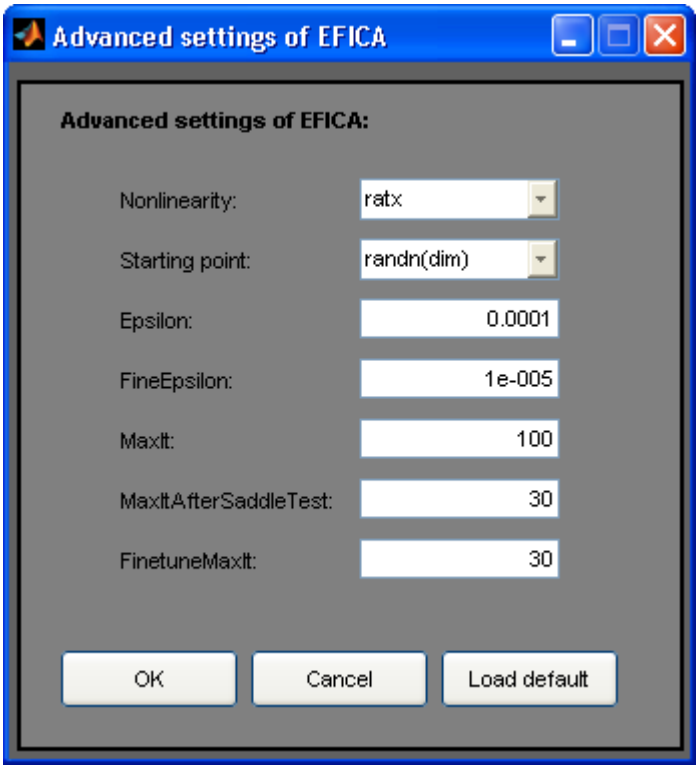
Výpočet následně spustíme kliknutím na tlačítko *Run*.

4.3.1 Nastavení EFICA

Mezi základní nastavitelné parametry algoritmu EFICA patří počáteční inicializace iteračního procesu (proměnná *ini*), volba typu nelinearity (proměnná *g*) a test sedlových bodů (*SaddleTest*). Kromě těchto vlastností však GUI umožňuje nastavovat ještě celou řadu jiných parametrů. Patří mezi ně *epsilon*, velmi malá hodnota, která zajišťuje, aby nedošlo k přerušení iterování, dokud jsou změny v odhadované demixovací matici po iteracích nad tuto mez. *Fineepsilon* se uplatní až v předposlední fázi algoritmu, která slouží ke zpřesnění výpočtu (tzv. finetuning). Zajišťuje v ní stejnou funkci jako parametr *epsilon*, hodnota *fineepsilon* by však měla být ještě menší. Hodnota proměnné *MaxIt* udává maximální počet

iterací. Pokud počet iterací přesáhne tuto mez, iterační proces skončí bez ohledu na to, zda je stop-kritérium menší než *epsilon*. Mez označená jako *MaxItAfterSaddleTest* se uplatní po fázi, kdy byl test sedlových bodů po skončení roven jedné (fáze pozitivní). V takovémto případě má pak smysl vykonat jen poměrně malé množství opravných iterací, proto je tato hodnota menší než *MaxIt*. Parametr *FinetuneMaxIt* se chová jako *MaxIt* pro předposlední fázi (finetuning).

Volbu *SaddleTestu* nastavujeme přímo v hlavním okně aplikace, ve stejném panelu jako výběr algoritmu. Ostatní parametry výpočtu zpřístupníme kliknutím na tlačítko „Advanced...“. Otevřeme tak novou *figure*, viz (obr. 4.3). Stisknutím tlačítka označeného jako „OK“ se jednotlivé zvolené hodnoty zapíší do workspace do proměnné *nastaveni*.



Obrázek 4.3: Rozšířené nastavení algoritmu EFICA

Abychom mohli bez problémů používat všechna tato detailní nastavení, musíme původní podobu konzolového programového předpisu algoritmu EFICA trochu pozměnit. Ten v hlavičce používá přímé přiřazení hodnot proměnných bez jakékoliv možnosti je z příkazové řádky měnit (viz následující kód).

```
epsilon=0.0001
fineepsilon=1e-5
```

```
MaxIt=100;
MaxItAfterSaddleTest=30;
FinetuneMaxIt=50;
```

Po malé úpravě umožníme tyto hodnoty uživateli pohodlně ovládat přímo z GUI, a to tak, že načteme globální proměnnou *nastaveni* z workspace a přiřadíme její jednotlivé položky vstupním hodnotám algoritmu.

```
global nastaveni

epsilon=nastaveni.epsilon;
fineepsilon=nastaveni.fineepsilon;
MaxIt=nastaveni.maxit;
MaxItAfterSaddleTest=nastaveni.maxitsaddle;
FinetuneMaxIt=nastaveni.EFICA;
```

SaddleTest může nabývat hodnot „true“ (test sedlových bodů bude proveden) nebo „false“ (test vypnut), u proměnné *g* máme na výběr mezi předvolenými hodnotami typů nelinearity (viz tab. 4.1).

rati	$g(x) = \frac{x}{1+\frac{x^2}{4}}$
ratx	$g(x) = \frac{x(2+\lvert x\rvert)}{(1+\lvert x\rvert)^2}$
tanh	$g(x) = \tanh(x)$
pow3	$g(x) = x^3$
gaus	$g(x) = x \cdot e^{-\frac{x^2}{2}}$

Tabulka 4.1: Přehled typů nelinearity

Položka „Starting point“ (*ini*) nabízí pouze matice „randn(dim)“ (pseudonáhodná matice o rozměrech dim x dim, kde dim představuje počet signálů vstupní proměnné – dle značení v předchozích kapitolách) a „eye(dim)“ (jednotková matice s rozměry dim x dim). Ostatní parametry mohou nabývat téměř libovolných číselných hodnot.

4.3.2 Nastavení WASOBI

Nastavitelnými parametry algoritmu WASOBI jsou v GUI (stejně tak jako v konzolové podobě) pouze *AR_order* a *rmax*. „AR order of separated sources" (*AR_order*), čili řád autoregresního procesu ekvivalentní počtu cros-korelačních matic, má vliv na přesnost výsledné separace. Parametr „Stabilization" (*rmax*), maximální povolená absolutní hodnota pólu, který je kořenem polynomu autoregresního procesu, může nabývat pouze hodnot z intervalu (0, 1). Zajišťuje odstranění nestability procesu, hodnota 1 znamená, že jsme tuto funkci vyřadili, její příliš malá hodnota naopak způsobí snížení přesnosti separace. Nejstabilnějšího výpočtu dosáhneme zadáním *rmax* = 0,99.

Oba tyto parametry načítá GUI přímo z komponent *EditText* těsně před spuštěním algoritmu (při volbě metody WASOBI z *Pop-Up Menu* se vzhled panelu přizpůsobí) a opět využívá stejné metody přenosu informace přes workspace. Jelikož GUI ovládá stejné množství podmínek výpočtu jako v běžné konzolové podobě, nemusíme do předpisu algoritmu nikterak zasahovat.

4.3.3 Nastavení COMBI, MULTI-COMBI A FCOMBI

Jak již víme, součástí výpočtu těchto metod jsou také algoritmy EFICA a WASOBI. Při návrhu jejich obsluhy jsem tedy nekonstruoval žádný zvláštní ovládací prvek. COMBI, MULTI-COMBI a FCOMBI využívají nastavení pouze svých dílčích algoritmů způsobem uvedeným v odstavcích 4.3.1 a 4.3.2.

4.4 Průběh programu a ukládání výstupů

Po stisknutí tlačítka *Run* umístěného v hlavním okně aplikace se spustí námi zvolený algoritmus dle předvolených detailních nastavení. GUI vyvolá dle potřeby konkrétní M-file s uloženým předpisem metody a v příkazovém řádku Matlabu se postupně zobrazují jednotlivé kroky výpočtu včetně oznámení o jeho délce. To se zapíše po jeho skončení jako poslední informace o stavu výpočtu.

K odhadu času potřebného k získání konečné podoby výstupů jsem použil časovač integrovaný přímo v Matlabu. Spouští se procedurou *tic* a je aktivní až do té doby, dokud v předpisu nenarazí na ukončovací příkaz *toc*. Do stejnojmenné proměnné pak načítá změřenou hodnotu v sekundách. Tu je vhodné zálohovat do proměnné vlastní, abychom s ní mohli dále pracovat. Ve svém GUI ji načítám do proměnné *t*, kterou pak vypisuji ve workspace na konci výpočtu každého algoritmu. Následující tabulka uvádí hodnoty času dosažené právě tímto způsobem (tab. 4.2). Takto získané údaje popisují pouze dobu zpracování jednotlivých dat v GUI. K získání těchto časů jsem použil náhodně vygenerovaná vstupní data (o rozměru 4x1000) zadáním příkazu

```
x = rand(4, 1000);
```

do příkazové řádky Matlabu. Získané časové hodnoty proměnné t (na nichž se také z velké části podílí výkon používaného počítače) tedy určují spíše poměrnou hodnotu mezi konkrétními použitými algoritmy.

Algoritmus	t [s]
EFICA	0,05 ÷ 0,06
WASOBI	0,045 ÷ 0,05
COMBI	cca 0,04
MULTI-COMBI	0,01 ÷ 0,025
FCOMBI	0,05 ÷ 0,07

Tabulka 4.2: Přehled rychlosti výpočtu použitých metod

Každá z metod má své specifické výstupy, i když jejich základ je pro všechny společný. Následuje jednoduchý stručný přehled výstupních proměnných řazených dle konkrétních použitých algoritmů.

EFICA

Demixovací matice: *Wefica* – výstupní *W* algoritmu EFICA
Wsymm – výstupní *W* algoritmu FastICA s testem (nebo bez) sedlových bodů

ISR matice: *ISRef* – pro složky EFICA
ISRsymm – pro složky FastICA

Ostatní proměnné: *status* – informuje o provedeném testu sedlových bodů (1 = test pozitivní, 0 = test negativní)

WASOBI

Demixovací matice: *Wwasobi* – odhadovaná demix. matice
Wsobi – počáteční odhad matice získané z FFDiag2

ISR matice: *ISR* – odhadovaná ISR matice

COMBI

Demixovací matice:	*W* – výsledná konečná demix. matice *Wefica* – odhad matice metodou EFICA *Wwasobi* – odhad matice metodou WASOBI
ISR matice:	*ISRef* – pro složky EFICA *ISRwa* – pro složky WASOBI
Ostatní proměnné:	*metoda* – označuje metodu, jejíž výsledek byl použit jako odhadnutá komponenta

MULTI-COMBI

Demixovací matice:	(viz *COMBI*)
ISR matice:	*ISRef1* – pro složky EFICA *ISRwa1* – pro složky WASOBI
Ostatní proměnné:	*nonseparablecomponents* – vícerozměrné komponenty, které již nelze dále separovat

FCOMBI

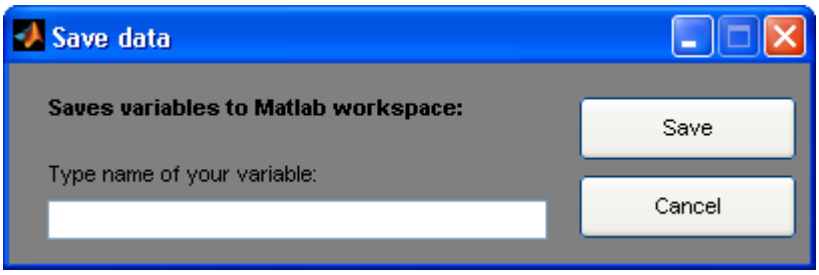
Demixovací matice:	*W* – odhadovaná matice W
ISR matice:	*ISR* – odhad matice ISR

Při aplikaci algoritmů pomocí mnou navrženého GUI však žádnou z těchto výstupních proměnných do workspace nenačteme. Jednotlivé předpisy algoritmů je sice do workspace zapíší, ale GUI je ihned po skončení výpočtu odstraní. Je to z toho důvodu, aby nedocházelo k přeplňování workspace přebytečnými proměnnými. Ne vždy si totiž uživatel přeje hned první výsledky metody ukládat či s nimi dále ve workspace pracovat. Pokud však uživatel dosáhne výsledků, které jej dostatečně uspokojují, navržený GUI nabízí možnost je uložit. Stiskem tlačítka „Save results“ v hlavním okně aplikace na středu horizontálně umístěného panelu vyvoláme okno viz (obr. 4.4).

Do pole *EditText* můžeme zadat vlastní „značku“, kterou GUI po stisku tlačítka „Save“ předřadí názvům standardních výstupních proměnných konkrétního algoritmu. To slouží ke zpřehlednění proměnných již uložených ve workspace. Značkami můžeme odlišit například výsledky několikanásobného

aplikování jednoho algoritmu s různými parametry. Třeba tím, že postupně předřazujeme ukládaným datům značky podle počtu opakování („pokus1_“, „pokus2_“ apod.). GUI pak ukládá jednotlivé proměnné dle následujícího příkladu: „pokus1_W“, „pokus1_ISR“,... . Nebo „pokus2_W“, „pokus2_ISR“... atd.

Tento způsob ukládání proměnných byl převzat z FastICA package.



Obrázek 4.4: Ukládání výsledků do workspace

4.5 Zobrazování vstupních a výstupních dat

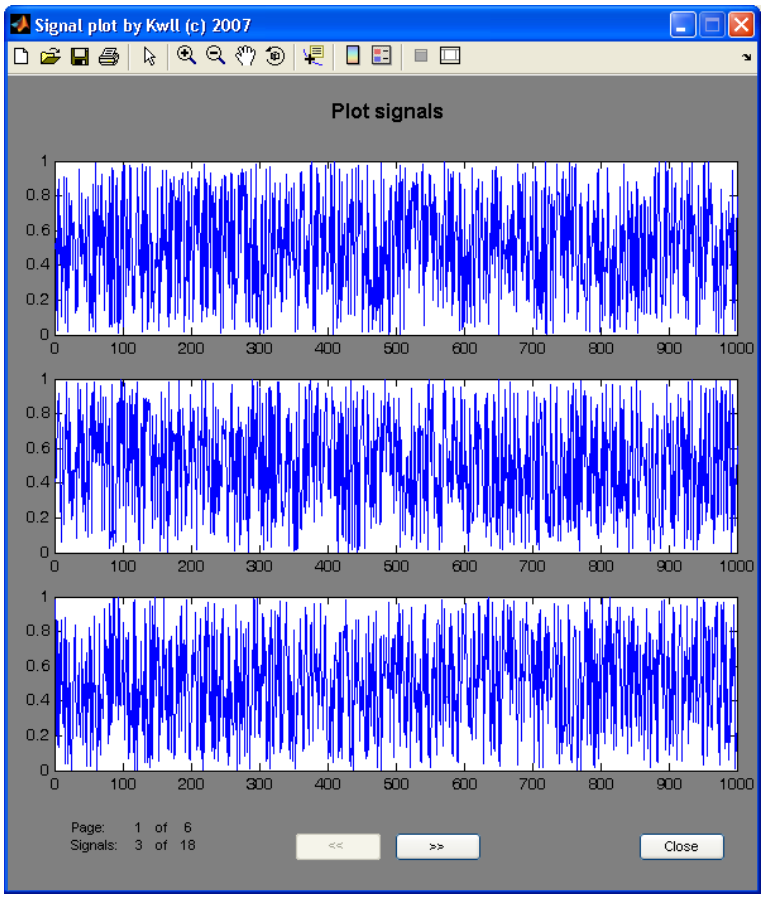
Navržený GUI pracuje se značným množstvím nejrůznějších signálů. Uživatel by tedy měl mít možnost si konkrétní data, ať už vstupní či výstupní, prohlédnout. Jednak to slouží k získání většího přehledu o podobě zpracovávaných dat a jejich okamžitou vizuální kontrolu, ale také větší představu o tom, co se s jakými signály přesně aplikováním algoritmů děje.

FastICA package, který byl mé práci vzorem, tuto možnost pochopitelně také podporuje, avšak z vlastní zkušenosti vím, že řešení zobrazování signálů není úplně optimální. Vykreslování dat probíhá pomocí příkazu *subplot*, přičemž jednotlivé signály tvoří matici o tvaru *dim* x 1 (sloupce x řádky), kde *dim* je rozměr zobrazovaných dat. Tento způsob shledávám použitelným pouze pro data do maximálního rozměru $dim = 5$. Pro signály $dim > 5$ pak dochází vlivem neoptimálního způsobu zobrazování k velkému zkreslení, pro data s počtem složek větším než deset se pak tento způsob stane naprosto nepoužitelným. Signály jsou pak od sebe naprosto neodlišitelné a při větším množství vykreslovaných složek se ve *figure* zobrazí pouze vodorovné osy s měřítky.

Navrhl jsem proto nový způsob zobrazovaní vstupních a výstupních dat. Stisk tlačítka „Plot data“ (pro vstupní) nebo „Plot results“ (pro výstupní) v hlavním okně aplikace vyvolá nový GUI pojmenovaný pracovně *myplot* (obr. 4.5).

Současně při tom zapíše do workspace proměnnou *myplot_signal*. *Myplot* ji načte do své vnitřní pracovní oblasti a proměnnou z workspace odstraní. *Myplot* umožňuje stránkování, na jednu stránku dovolí zobrazit maximálně tři signály, nemůže tedy dojít k jejich překrytí. Pomocí takto navrženého zobrazování můžeme pohodlně nechat vykreslit libovolné množství signálů (data s libovolným

rozměrem, počtem složek), pohybovat se mezi jednotlivými stránkami můžeme snadno stiskem příslušných tlačítek. Pozici konkrétní prohlížené stránky včetně informace o celkovém počtu signálů k zobrazení uživatel nalezne ve spodní levé části programu.



Obrázek 4.5: Zobrazení vstupních/výstupních dat

Figure tvořící základ celého okna *myplot* jsem ponechal nástrojovou lištu, která uživateli umožní aspoň minimální práci se zobrazovanými signály, jako např. přibližování, posunování, 3D rotaci, tisk, ukládání jejich grafické podoby atd.

Myplot je prozatím zprovozněn tak, aby fungoval pouze jako součást hlavního GUI. Využívá předem předdefinované a pevně stanovené proměnné z workspace, přičemž uživatel nemá možnost je žádným způsobem měnit. Pokud se však *myplot* časem osvědčí, je navržen tak, aby jen po drobných úpravách zdrojového kódu fungoval jako univerzální prostředek pro zobrazování většího počtu signálů.

4.5.1 Vykreslení matice ISR

GUI nabízí vykreslování matic ISR, k jejichž odhadu dochází při každé aplikaci algoritmu. Pokud ji chce uživatel zobrazit, docílí toho kliknutím na tlačítko „View ISR“ umístěné v hlavním okně aplikace.

Základem vykreslování matice je procedura *showmatrix*. Ta zobrazuje libovolnou vstupní matici pomocí šedého spektra, jednotlivým prvkům přiřazuje odstín odpovídající jejich vztažné hodnotě k ostatním prvkům matice. Procedura načte vstupní hodnoty matice, největšímu prvku přiřadí nejtmavší barvu (černá) a nejmenšímu nejsvětlejší (bílá). Barvy ostatních prvků odpovídají rovnoměrnému rozložení poměrných odstínů dle velikosti jejich hodnoty.

Matice ISR je však výhodné pro lepší názornost zobrazovat v jednotkách *dB*. Převedení na jednotky *dB* provedeme příkazem

```
A = 10*log10(A);
```

Z definice matice ISR plyne, že na diagonále budeme často vyžadovat logaritmus nuly, což je nepřípustné z důvodu následné chybné interpretace diagonálních prvků matice ISR. Tento jev eliminujeme následujícím způsobem. Nejprve zavedeme pomocnou matici *C*, která bude mít všechny nediagonální prvky společné s maticí *A*, na diagonále pak samé 1 (z důvodu $\log_{10} 1 = 0$).

```
[sloupce radky] = size(A);
for i = 1:sloupce
        for j = 1:radky
                if i == j
                        C(i,j) = 1;
                else    C(i,j) = A(i,j);
                end
        end
end
```

Tuto matici následně převedeme na *dB* a určíme minimální hodnotu zlogaritmovaných prvků matice *CdB*

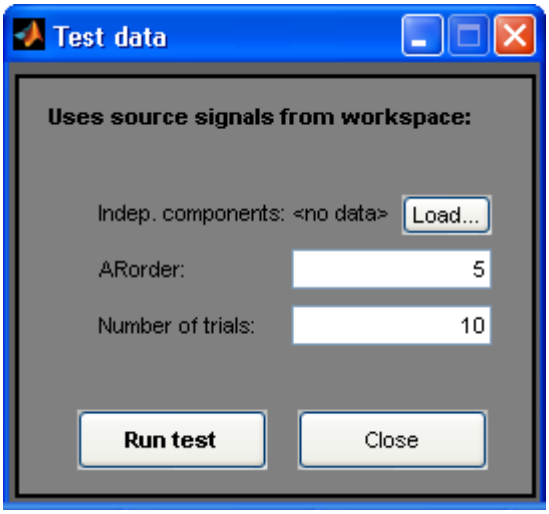
```
CdB = 10*log10(C);
hranice = min(CdB(:));
```

Tato minimální hodnota tvoří hranici, kterou nepřesáhne žádný z ostatních prvků matice *CdB*. Pokud vyjdeme z tohoto předpokladu, pak pro grafickou odlišitelnost diagonálních prvků (které jsou vždy limitně rovny $-\infty$) postačí, když jejich hodnota bude ve vztahu k ostatním vždy menší, než hodnota takto stanovené hranice. Pak může následovat kód, který převede prvky matice *A* na hodnoty v *dB*. Tam, kde je prvek roven 0 (diagonála), nahradí je hodnotou *kr*-krát větší, než je hraniční hodnota. Koeficient *kr* představuje konstantu rozlišitelnosti a empiricky

bylo zjištěno, že jeho optimální hodnota se pohybuje v intervalu $\langle 1.2, 1.5 \rangle$.

```
for i = 1:sloupce
        for j = 1:radky
                if A(i,j) > 0
                        B(i,j) = 10*log10(A(i,j));
                else B(i,j) = kr*hranice;
                end
        end
end
A = B;
```

Takový zápis je poměrně rozsáhlý na to, aby byl aplikován před každým (ve zdrojovém kódu GUI poměrně častým) voláním procedury *showmatrix*. Ta byla proto pozměněna na proceduru *showmatrixISR*, která již tuto část kódu obsahuje.



Obrázek 4.6: Testování dat

4.6 Testování dat

GUI nabízí pro metody EFICA a WASOBI možnost testování dat. To může sloužit buď k ověření správnosti separace nebo k ověření vlastností separovaných signálů. V obou případech jsou výsledkem testu dvě podoby matice ISR, k jejichž odhadu dochází na sobě nezávislými způsoby. Porovnáváme teoretickou ISR (matice počítaná z odseparovaných dat na základě teoretické analýzy příslušného algoritmu) s maticí získanou empiricky. Empirickou matici získáme tak, že vytvoříme repliky nezávislých původních signálů (tzv. bootstrap). Zamixujeme je náhodně generovanou mixovací maticí A a zamíchané je opět rozseparujeme.

Výsledná gain matice je bootstrap realizací matice empirické.

Vstupními daty jsou nezávislé signály. Ty můžeme získat například separováním ze zamixovaných dat prostřednictvím hlavního okna GUI nebo můžeme použít nezávislé signály vlastní (známé i neznámé).

Testovací oblast vyvoláme stiskem tlačítka „Test data...“ umístěného v hlavním okně GUI. Tím spustíme okno viz (obr. 4.6). Testování probíhá v několika základních krocích. Tím prvním je načítání dat. Toho dosáhneme stiskem tlačítka „Load...“, které vyvolá stejný dialog jako při načítání globálních vstupních signálů v hlavním okně aplikace, viz (obr. 4.2). Dalším krokem je stanovení základních podmínek chodu testu. Vstupními proměnnými jsou (kromě nezávislých signálů) také *ARorder* (řád AR procesu – pouze pro metodu WASOBI) a *trials* (počet opakování testu – pro obě metody). Po stisku tlačítka „Run test“ GUI načte vstupní hodnoty z workspace do vlastních lokálních proměnných.

Pro metodu EFICA provedeme pouze bootstrap. Pro každou složku nezávislých dat *s*(i,:) získáme náhodnou permutací jejich realizací signál *shat*(i,:) se stejnými vlastnostmi jako původní

```
s_r = s(i,:);
shat(i,:) = s_r(randperm(length(s_r)));
```

Pro metodu WASOBI je snahou vytvořit realizaci signálu se stejnými vlastnostmi v rámci AR modelu. Nejprve odhadneme koeficienty AR procesu každého signálu nezávislých dat

```
coefest = lpc(s(i,:),ARorder);
```

Na základě odhadů AR modelu vypočteme inovační proces. Novou realizaci inovačního procesu *y* vytvoříme opět pomocí bootstrap, tedy permutací získáme novou inovaci *shat*(i,:) se stejnými vlastnostmi jako původní a nový signál vytvoříme zpětnou filtrací nového inovačního procesu, čímž vznikne signál se stejným spektrem jako měl původní. Tento způsob je ovšem hodně citlivý na předpoklady, které musí původní signál splňovat.

```
s_r = s(i,:);
y = filter([0 -coefest(2:end)],1,s_r);
y = y(randperm(length(y)));
shat(i,:) = filter(1,coefest,y);
```

Těmito způsoby tedy získáme nová vstupní data *shat*, se kterými budeme dále pracovat. Jak již víme, pro lineární model bez paměti platí $X = A \cdot S$. Matici A vygenerujeme příkazem

```
A = rand(dimenze_dat);
```

a vynásobíme ji daty *shat*. Vyrobíme tím zamixovaná data *x*, na která GUI aplikuje algoritmus předem zvolený v hlavním okně aplikace. Následuje odhad gain matice a permutační matice.
Matici *ISRemp* získáme v Matlabu předpisem

```
ISRemp = (G - diag(diag(G))).^2./(diag(G).^2*ones(1,dim));
```

kde *G* představuje gain matici, *dim* je dimenze vstupních dat. Teoretickou ISR, jež je výsledkem příslušného algoritmu, pak musíme zleva i zprava vynásobit výše získanou permutační maticí. Celý tento postup zopakujeme *trials*-krát a z výsledných matic provedeme aritmetický průměr.

Empirická matice je odhadnutá s neznámou odchylkou, jelikož nemáme k dispozici skutečné repliky signálů (pouze bootstrap). To může způsobit některé nepřesnosti (odchylky) empirické a teoretické matice.

Konečnou fází testu je úplné vykreslení matic ISR dle odstavce 4.5.1.

4.6.1 Simulace testování

Popsaný způsob testování ověříme na datech, která splňují všechny teoretické předpoklady. V testu byly použity defaultní hodnoty (*ARorder* = 5, *trials* = 10).

Pro metodu EFICA použijeme v prvním případě vstupní signály se strukturou

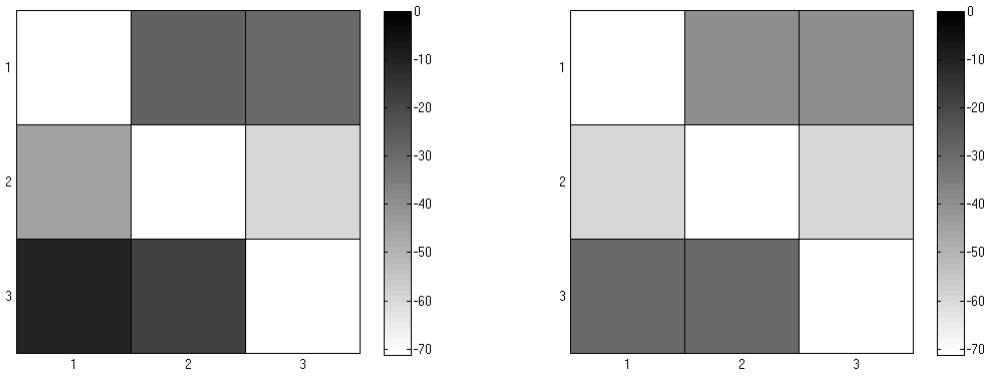
```
s(1,:) = rand(1,1000);          (rovnoměrné rozložení)
s(2,:) = sign(randn(1,1000));   (binární rozložení)
s(3,:) = randn(1,1000);         (gaussovo rozložení)
```

Výsledkem testu jsou empirická (obr. 4.7a) a teoretická (obr. 4.7b) ISR matice. Data obsahují pouze jeden signál s gaussovým rozložením, jsou tedy dobře odlišitelná. To potvrzuje i výsledek testu.

Druhým případem bude situace, kdy do složky vstupních dat *s* přidáme další signál s gaussovým rozložením.

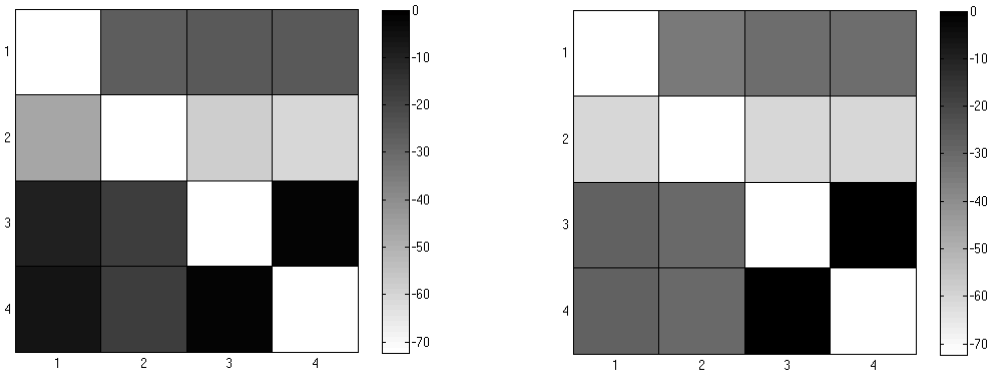
```
s(4,:) = randn(1,1000);
```

Z teorie víme, že pokud data obsahují dva nebo více signálů s gaussovým rozložením, jsou potom neodseparovatelné. To můžeme pozorovat na (obr. 4.8), který tuto skutečnost potvrzuje. Vidíme, že signály 1 a 2 jsou odseparovatelné vzájemně i od všech ostatních, signály 3 a 4 jsou odseparovatelné od ostatních, však od sebe nikoliv (nejtmavší pole). Opět můžeme tvrdit, že oběma způsoby získávání matice ISR dosáhneme přibližně stejných výsledků.



a) b)

Obrázek 4.7: Empirická (a) a teoretická (b) ISR matice
metoda: EFICA, případ 1 (3 signály, z toho jeden s gaussovým rozložením)



a) b)

Obrázek 4.8: Empirická (a) a teoretická (b) ISR matice
metoda: EFICA, případ 2 (4 signály, 2 s gaussovým rozložením)

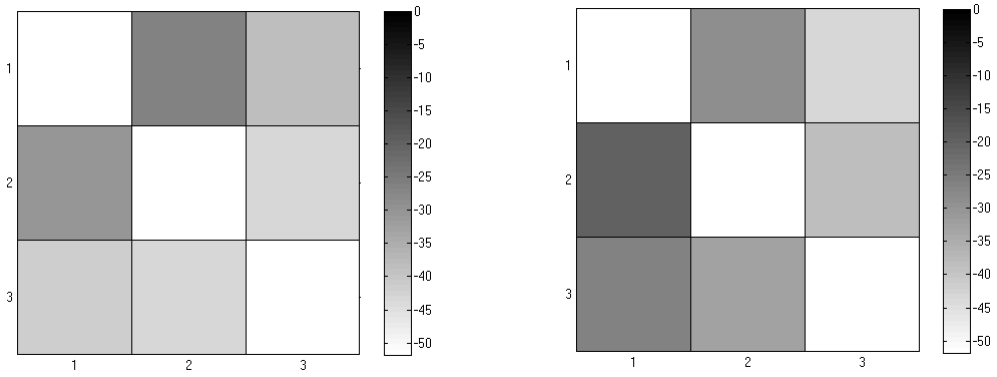
Pro algoritmus WASOBI je vhodné vytvořit data následujícím způsobem

```
s(1,:) = filter(b1,a1,randn(1,1000));
s(2,:) = filter(b2,a2,randn(1,1000));
s(3,:) = filter(b3,a3,randn(1,1000));
```

Pokud zvolíme koeficienty filtru tak, aby spektra jednotlivých signálů byla co nejvíce odlišná, dosáhneme přesné separace všech složek vstupních dat. O tom se

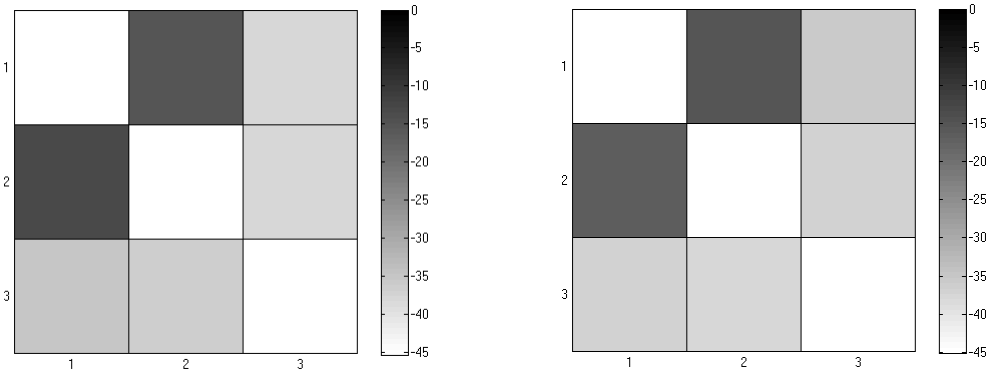
můžeme přesvědčit na (obr. 4.9), kde vidíme ISR matice dat generovaných příkazy

```
s(1,:) = filter(1,[1 -0.4],randn(1,1000));
s(2,:) = filter(1,[1 -0.7],randn(1,1000));
s(3,:) = filter(1,[1 0.7],randn(1,1000));
```



a) b)

Obrázek 4.9: Teoretická (a) a empirická (b) ISR matice
metoda: WASOBI, případ 1 (spektra signálů odlišná)



a) b)

Obrázek 4.10: Empirická (a) a teoretická b) ISR matice
metoda: WASOBI, případ 2 (spektra 2 signálů stejná)

Zvolíme-li koeficienty filtrů tak, aby spektra dvou signálů byla totožná, ověříme, že v takovém případě jsou signály metodou WASOBI neodseparovatelné.

```
s(1,:) = filter(1,[1 -0.7],randn(1,1000));
s(2,:) = filter(1,[1 -0.7],randn(1,1000));
s(3,:) = filter(1,[1 0.7],randn(1,1000));
```

I tentokrát dostaneme stejný výsledek jak teoretickým, tak empirickým odhadem matice ISR. Na (obr. 4.10) vidíme, že signály 1 a 2 jsou od 3. signálu odseparovatelné, avšak mezi sebou nikoliv. Signál 3 je odlišitelný od obou zbývajících signálů.

4.7 Shrnutí

V konečné, poslední fázi byla celá aplikace z mé strany podrobena relevantním testům. Pokusil jsem se nasimulovat nestandardní situace, ke kterým může za jistých podmínek při používání programu dojít. Výsledkem těchto testů bylo vyladění aplikace tak, aby splňovala veškeré požadavky, které na ni byly kladeny. GUI je dle mého názoru plně provozuschopný. Kompletní popis funkcí, které ve své práci uvádím, beze zbytku ovládá.

V GUI byla také ošetřena tlačítka. To znamená, že pokud stiskneme „Load data“ dvakrát po sobě, vyvoláme pouze jedno dialogové okno pro načítání proměnných, nikoliv dvě. Takto se chovají všechna tlačítka, která tuto drobnou úpravu z uživatelského hlediska vyžadují. GUI dále uživateli nedovolí např. vyvolat figure pro ukládání výsledků separace, pokud nebyl aplikován žádný z algoritmů. Ten pak nejde použít, dokud nenačteme žádná vstupní data apod. Veškeré tyto skutečnosti oznamuje GUI v příkazovém řádku Matlabu pomocí funkce *disp*.

Z dodatečných grafických úprav, které mají uživateli napomáhat v práci s programem, mohu např. jmenovat zvýraznění nejpodstatnějších funkčních tlačítek fontem s vlastností „Bold“ či barevnou signalizaci, která uživatele zcela přirozeným způsobem provádí během práce s aplikací.

5 Závěr

Úspěšně byly vypracovány veškeré body uvedené v zadání bakalářské práce. Nastudoval jsem teorii slepé separace signálů, jak její principiální stránku, tak použití některých konkrétních algoritmů v praxi. Seznámil jsem se též s nástroji sloužícími k vytváření grafického uživatelského rozhraní v Matlabu a naučil jsem se je používat k návrhu konkrétních v praxi využitelných aplikací. Navrhl jsem plně funkční kompaktní GUI k ovládání algoritmů EFICA, WASOBI, COMBI, MULTI-COMBI a FCOMBI. Rozhraní umožňuje zobrazování vstupních (zamixovaných) i výstupních (odseparovaných) signálů a uložení výstupních proměnných algoritmů do workspace pro jejich případné další použití.

Reference

[1] The FastICA package for MATLAB
[http://www.cis.hut.fi/projects/ica/fastica/]

[2] The MathWorks, *Getting started with MATLAB® 7*, září 2006
[http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf]

[3] The MathWorks, *Creating Graphical User Interfaces*, září 2006
[http://www.mathworks.com/acces/helpdesk/help/pdf_doc/matlab/buildgui.pdf]

[4] The MathWorks, *The Mathworks Helpdesk*
[http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_product_page.html]

[5] Hyvärinen, A. and Oja E., *Independent component analysis: A Tutorial*, 1999

[6] Hyvärinen, A. and Oja E., *A fast fixed-point algorithm for independent component analysis*, 1997

[7] Cardoso J.-F., *Blind signal separation: statistical principles*, 1995

[8] Koldovský, Z., *Analýza nezávislých komponent – odhad vzájemné informace a separace konvolutorních směsí*, výzkumný úkol, 2001

[9] Koldovský, Z., *Analýza nezávislých komponent, FastICA a přímý výpočet vzájemné informace*, rešeršní práce, 2000

[10] Koldovský, Z., *Fast and Accurate Methods for Independent Component Analysis*, Ph.D. Thesis, 2005